



> Конспект > 7 урок > Модели для работы с текстом, введение в эмбединги. FASTTEXT, DSSM

> Оглавление

> Оглавление

> Word2Vec

> Пример, CBOW

> Пример, Skip-gram

> Формирование тренировочного набора

Распределение по словарю

> Смена метода обучения

Пример

Негативное семплирование

> Эмбединги w2v

Минусы w2v

Пример с рекуррентной сетью

> FastText by Facebook

Пример

> Хэширование словаря

> Deep Structured Semantic Models (DSSM)

> Convolutional Deep Structured Semantic Models (C-DSSM)

> MatchPyramid

> Аналогия со сверточными сетями

> Комбинирование методов и улучшение архитектур

> [DeepRank](#)

> [Conv-KNRM](#)

> [Резюме](#)

> Word2Vec

Это алгоритм, решающий задачу языкового моделирования.

Можно обучать на любых текстах. Чем больше корпус — тем лучше.

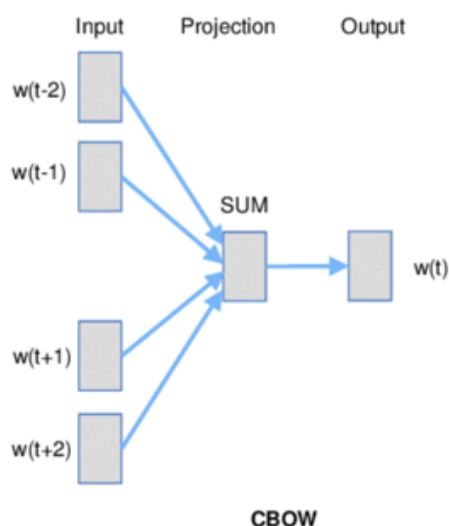
Основная идея — схожие слова встречаются в схожих контекстах.

Метод обучения:

1. Сформировать словарь (с учётом предобработки с помощью, например, [стемминга](#) или [лемматизации](#)).
2. Двигаться "скользящим окном по тексту", формируя контекст слова ("соседей").
3. Учиться предсказывать [распределение по словарю](#).

> Пример, CBOW

CBOW (*Continuous Bag of Words*) предсказывает текущее слово, исходя из окружающего его контекста.



Предложение: "Какое-нибудь достаточно длинное и распространённое предложение, выступающее в качестве примера".

Ширина окна равна 5.

"Предложение" — центральное слово, которое нужно предсказать, а два слова справа и слева — это его контекст.

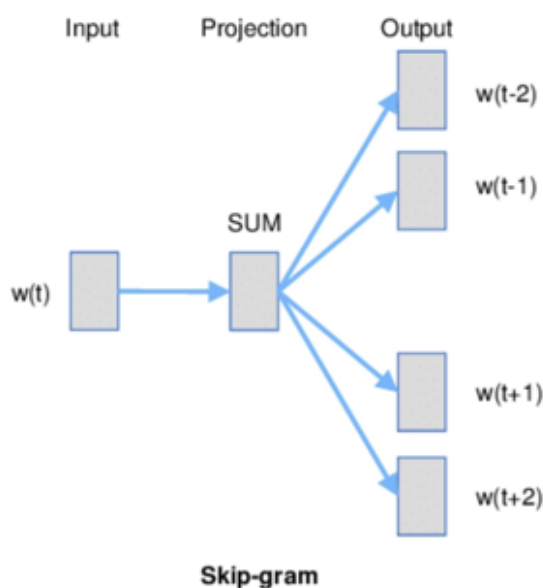
Предобработав слова контекста, можем записать условную функцию от них:

$F(\text{и, разнообраз, выступа, в}) \approx \text{предложение}$

Задача машинного обучения в данном случае состоит в том, чтобы получить такую модель F , которая выдаёт для входных признаков в виде указанных в скобках слов выходные значения, характеризующие ожидаемое слово. В данном случае это слово "предложение".

> Пример, Skip-gram

Похож на CBow, однако обладает ровно противоположным смыслом: такая модель должна предсказывать контекст по слову, т.е. на вход в качестве признаков в обучаемую функцию подаётся центральное слово из окна, а на выходе ожидается указание на то, какими должны быть слова из контекста.



Предложение: "Ещё одно креативное предложение, но уже для иного примера (который справа)".

Ширина окна равна 5.

Теперь центральное слово становится фичей, а жёлтые слова необходимо предсказать.

$F(\text{предложение}) \approx [\text{один, креатив, но, уже}]$

> Формирование тренировочного набора

Тренировочный набор:

Input = [и, разнообраз, выступа, в]

Target = предложение

Тренировочный набор для CBOW

Для метода **Skip-gram** обычно **формируют пары**, пытаюсь по слову за раз предсказать одного соседа из контекста, а не сразу всех.

Тренировочный набор:

Input = предложение ; Target = одно

Input = предложение ; Target = креативное

Input = предложение ; Target = но

Input = предложение ; Target = уже

Тренировочный набор для Skip-gram

В качестве целевого значения и в одном, и в другом методе выступает лишь одно слово из словаря.

Каким образом модель может выбрать такое слово? Самый простой и понятный способ — **предсказание вероятности для каждого слова** из словаря. То **слово, для которого вероятность выше**, считается наиболее подходящим. Это и есть "предсказание распределения по словарю".

Распределение по словарю

	Предсказания	Таргет
абзац	0.01	1
август	0.12	0
агроном	0.17	0
...
язва	0.01	0
ящерица	0.02	0

Пример распределения

Для каждого слова ответ модели — это величина от 0 до 1, указывающая на вероятность встретить слово в некотором контексте, поступившем на вход. Так как в скользящем окне мы знаем, какое действительно слово ожидается, то можно составить вектор ответов, с которым будут сравниваться предсказания — это **oneHot**-вектор, в котором **одна единица** выставлена в индексе, что соответствует расположению ожидаемого слова в словаре.

По разности целевого и предсказанного вектора, задающего распределение вероятности по словам из словаря, можно:

1. Рассчитать ошибку модели;
2. Посчитать градиенты;

3. Сделать шаг градиентного спуска для изменения параметров с целью уменьшения ошибки.

> Смена метода обучения

Проблема — такая модель слишком медленная.

Решение — перейти от решения задачи предсказания вероятностей слов из словаря к задаче бинарной классификации того, является ли слово соседом или нет. При этом логит для определения вероятности формируется простым перемножением векторов основного слова и слова контекста.

Создаётся две матрицы одинаковых размеров, которые инициализируются случайным образом:

- Высота матрицы — количество слов в словаре;
- Ширина — размерность эмбединга.

Одна матрица отвечает за эмбединги слов, другая — за контекст.

Пример

Берём значение эмбединга (emb) для слова из центральной части окна, из выборки, которую формировали выше, т.е. эмбединг слова "предложение".

Далее из второй матрицы берём эмбединг слова из контекста (emb_c).

$$\text{sigmoid}(emb(\text{предложение}) \cdot emb_c(\text{один})) \rightarrow 1$$

$$\text{sigmoid}(emb(\text{предложение}) \cdot emb_c(\text{отсебятина})) \rightarrow 0$$

В результате скалярного произведения этих векторов получается некоторое число. Логистическая функция от этого числа должна стремиться к 1, если слова стоят рядом, т.е. одно образует контекст другого.

Негативное семплирование

Можно заметить, что выборка собиралась таким образом, что в датасете представлены только положительные пары. Поэтому важно добавить в выборку негативные примеры. Такая техника называется `negative sampling` и самый простой её пример — это случайный выбор слов из словаря.

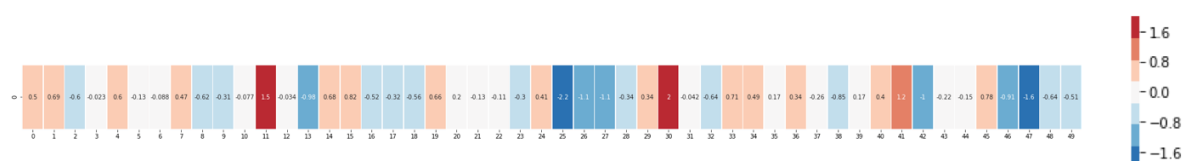
Поскольку словарь огромный, то, скорее всего, будет выбрано слово не из реального контекста. Да, иногда будут происходить коллизии, подающие

шумный сигнал модели, но это большая редкость — на это можно не обращать внимания.

Для такого "случайного" слова таргет устанавливается равным 0, и таким образом происходит обучение: слова, которые встречаются часто в одном и том же контексте, т.е. максимально похожи на эмбединги контекстных слов, должны иметь очень похожие вектора.

> Эмбединги w2v

Ещё больше картинок: [здесь](#).



Предобученные на английской Википедии эмбединги размерности 50

Видно, что есть ярко выраженные компоненты (большие по модулю), и есть околонулевые компоненты вектора бледного цвета.

Сравним раскрашенные эмбединги для трех слов: король, мужчина и женщина.

“king”



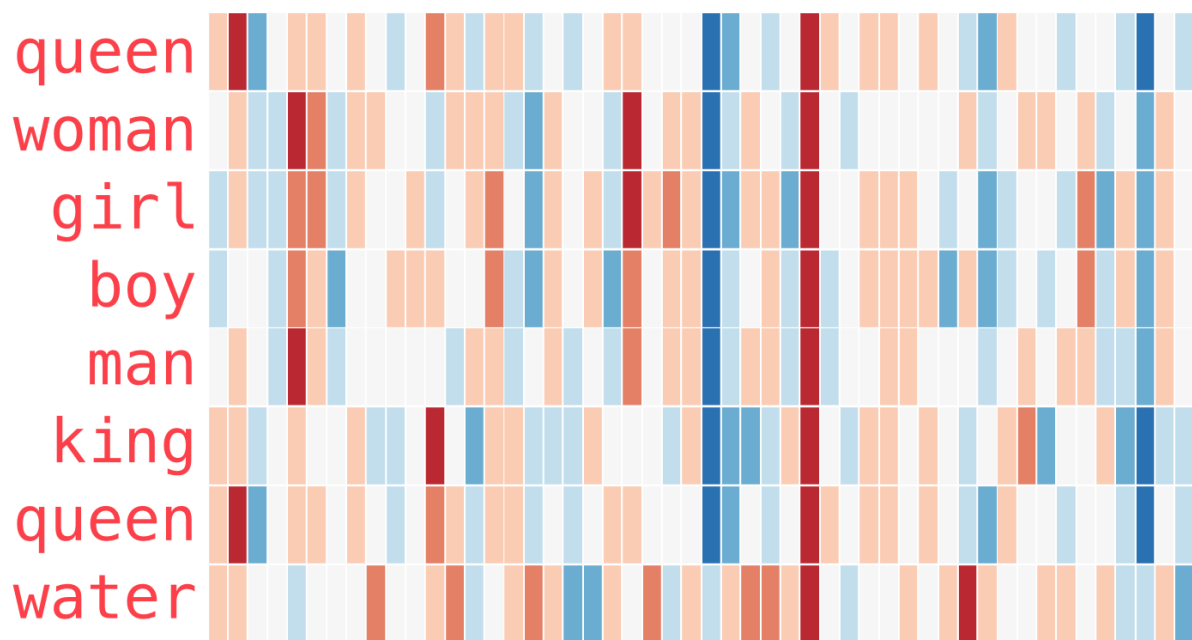
“Man”



“Woman”



«Мужчина» и «женщина» гораздо больше похожи друг на друга, чем на верхний вектор для слова «король»



Более показательный пример сравнения бОльшего количества слов

В примере можно заметить несколько особенностей:

- Наличие **общей колонки красного цвета** чуть правее центра. Это означает, что у всех слов ("королева", "женщина", "девочка", "мальчик", "мужчина", "король", "вода") есть что-то общее, но мы не знаем, что именно. Можно предположить, что это язык (в данном случае английский), ведь при обучении в корпус могли попасть латынь или другие языки, которые встречаются в определённом контексте.
- Женщина и девочка **попарно схожи** примерно так же, как и мальчик и мужчина.
- Через все слова, кроме последнего, проходит **вертикальная синяя линия** по центру, означающая отрицательное значение эмбединга в этой пространственной компоненте. Слово "вода" единственное неодушевленное, которое не представляет человека. Можно предположить, что именно такое разделение заложено в этой части эмбединга.

king - man + woman \approx queen



Для таких векторов даже работает математика.

Эмбединги слов можно использовать для многих задач, в частности для **матчинга** (DRMM).

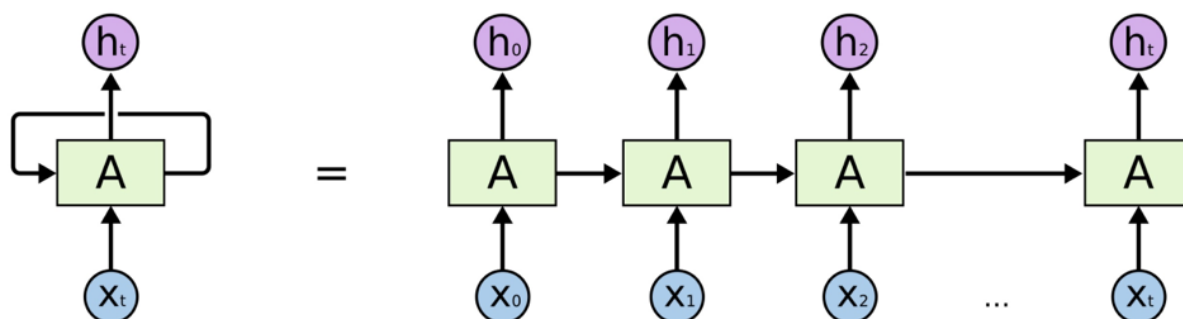
Существует множество способов **агрегации информации**: от простых вроде усреднения до BiLSTM.

Если доступна возможность **дообучать эмбединги**, то можно получать выгоду от **transfer learning**.

Минусы w2v

- Не учитывается морфология слов;
- Слова вне словаря (*Out Of Vocabulary*, **oov**) не обрабатываются.

Пример с рекуррентной сетью



Пример агрегации текста из T слов с помощью рекуррентного блока

Допустим, решается задача классификации текстов по категориям новостей: политика, спорт и т.п. Текст новости состоит из t слов, для которых существуют уже предобученные эмбединги.

Возьмём эмбединг первого слова X_0 , подадим его в рекуррентный блок, который применит некоторые **нелинейные преобразования** к эмбедингу и выдаст нечто — **скрытое состояние** (*hidden state*). Сейчас он несёт в себе только смысл одного слова.

Hidden подаётся дальше. Теперь уже в рекуррентный блок подаётся **скрытое состояние после первого слова вместе с эмбедингом второго слова** X_1 .

Полученное состояние будет характеризовать два слова.

Повторив эту операцию для всех слов из текста, получим некоторое общее представление о том, что вообще было написано в тексте. Это лучше простого усреднения, потому что появляется выучиваемая возможность **не учитывать определённые слова**, например союзы, предлоги, очень частые слова, **stop-слова**, которые при усреднении размывали бы общий смысл и делали бы его менее выраженным.

После прохода RNN по всему тексту финальный Hidden используется в качестве признаков для модели классификации, и решается финальная задача определения категории. Такой вектор может использоваться для задачи ранжирования текстов общего вида и **слабо применим к матчингу**, так как большая часть информации об уникальных особенностях конкретного товара всё равно теряется.

> FastText by Facebook

Ознакомиться со статьёй можно [здесь](#).

Все слова разбиваются на **N-граммы**. Эмбединг слова равен сумме:

- **Эмбединга** самого **слова** (если есть в словаре);
- Среднему **всех эмбедингов N-грамм** в слове.

Пример

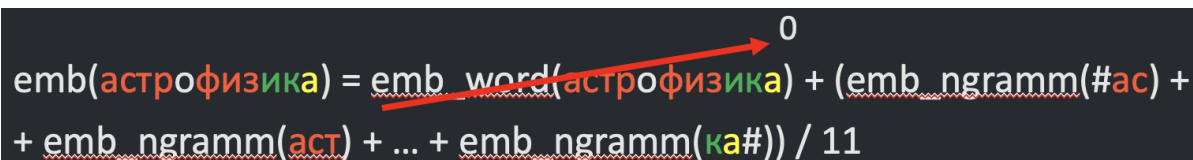
Для удобства цветами в слове выделены разные составляющие: **корни**, **суффикс**, **окончание**.

Возьмём слово **астрофизика**.

Добавим символы начала и конца слова: #астрофизика#

Разобьём слово на триграммы ($N = 3$):

#ас, аст, стр, тро, роф, офи, физ, изи, зик, ика, ка# — получаем 11 триграмм.


$$\text{emb}(\text{астрофизика}) = \text{emb_word}(\text{астрофизика}) + (\text{emb_ngramm}(\text{\#ас}) + \text{emb_ngramm}(\text{аст}) + \dots + \text{emb_ngramm}(\text{ка\#})) / 11$$

Первое слагаемое превращается в 0, так как, скорее всего, такого слова не будет в словаре, но зато будут все триграммы.

Плюс такого подхода заключается ещё и в том, что триграмм для заданного алфавита счётное множество, и их количество куда меньше, чем количество слов в языке.

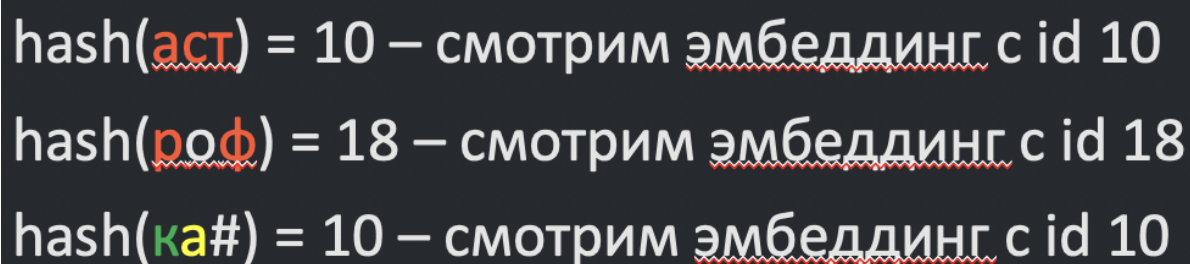
Более того, отдельные словоформы могут восприниматься без потери смысла: достаточно выучить, что суффикс "очк" — уменьшительно-ласкательный, передаёт оттенок нежности, и что слово "мама" — это родитель женского пола. Тогда слово "мамочка" не поставит в тупик модель, даже если слово ранее не встречалось. Такова **интуиция**, стоящая за разбиением на N-граммы.

> Хэширование словаря

Проблемы:

- Несмотря на то, что N-грамм меньше, чем слов в языке, всё ещё остается **возможность встретить редкие сочетания**;
- Желание **уменьшить количество параметров** модели никуда не делось.

Было предложено ограничить словарь N-грамм каким-нибудь размером и использовать **хэш-функцию** от N-граммы в качестве указателя на то, какому id она соответствует.



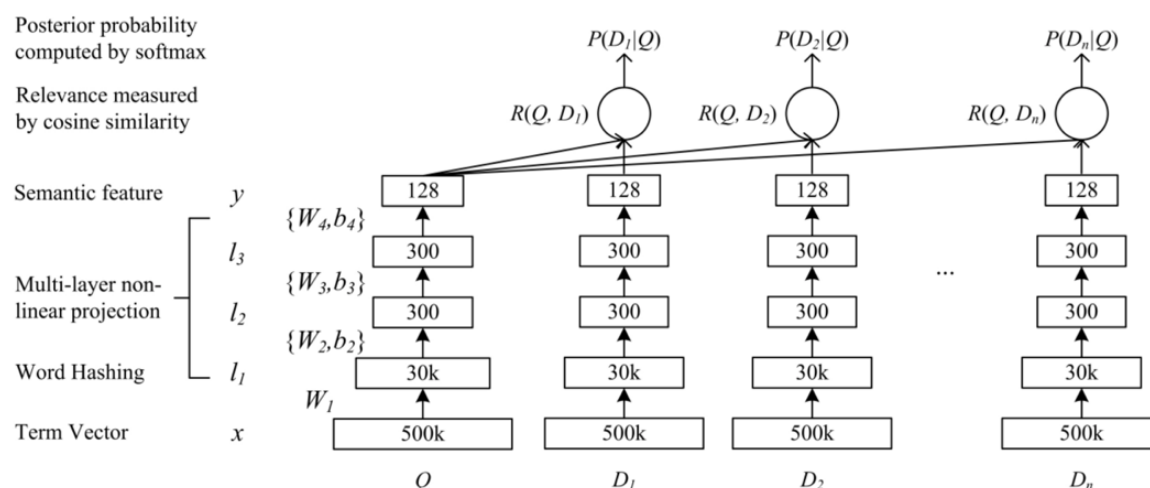
hash(аст) = 10 — смотрим эмбединг с id 10
hash(роф) = 18 — смотрим эмбединг с id 18
hash(ка#) = 10 — смотрим эмбединг с id 10

Пример с астрофизикой

Несколько разных N-грамм **могут оказаться с одинаковым id**, и это **нормально**. При работе с хэш-функциями такие события называются "**коллизиями**". Проверяется схожесть не отдельных N-грамм и вероятность их попадания в один бин хэш-таблицы с эмбедингами, а вероятность того, что два разных слова будут заданы одним набором id.

> Deep Structured Semantic Models (DSSM)

В компании Майкрософт специалисты проверили ([читать](#)), возможно ли на реальных данных о кликах в поисковой системе от начала и до конца обучить модель, формирующую эмбединги, релевантные задаче информационного поиска? И оказалось, что возможно.



Архитектура решения — уже знакомый вам подход с двумя отдельными ветвями сети для запросов и документов, которые могут иметь как одинаковые, так и разные веса.

Каждая из ветвей обрабатывает текст, разбивая слова на N-граммы, и векторизует его в **representation-focused** манере с помощью нескольких полносвязных слоёв с нелинейностями между ними.

Основа эмбедингов здесь фактически полностью аналогична **FastText** — слова кодируются N-граммами с применением хэширования и проецируются сначала в пространство крайне высокой размерности (30000) и постепенно сужаются к выходному вектору размерности 128.

Учится вся система как **бинарный классификатор**: кликали на определённый документ D при выдаче его в ответ на запрос Q или нет.

Вероятность определяется логистической функцией от логита, выраженного косинусной схожестью между векторами запроса и документа:

$$R(Q, D) = \text{cosine}(y_Q, y_D) = \frac{y_Q^T y_D}{\|y_Q\| \|y_D\|}$$

DSSM были представлены в 2013 году, но их активно используют и по сей день.

Минусы подхода:

- Теряется информация о порядке слов;
- Теряется информация о частоте употребления слова.

> Convolutional Deep Structured Semantic Models (C-DSSM)

Для исправления проблем DSSM было решено применить свёртки по некоторому скользящему окну.

Semantic layer: y

Affine projection matrix: W_s

Max pooling layer: v

Max pooling operation

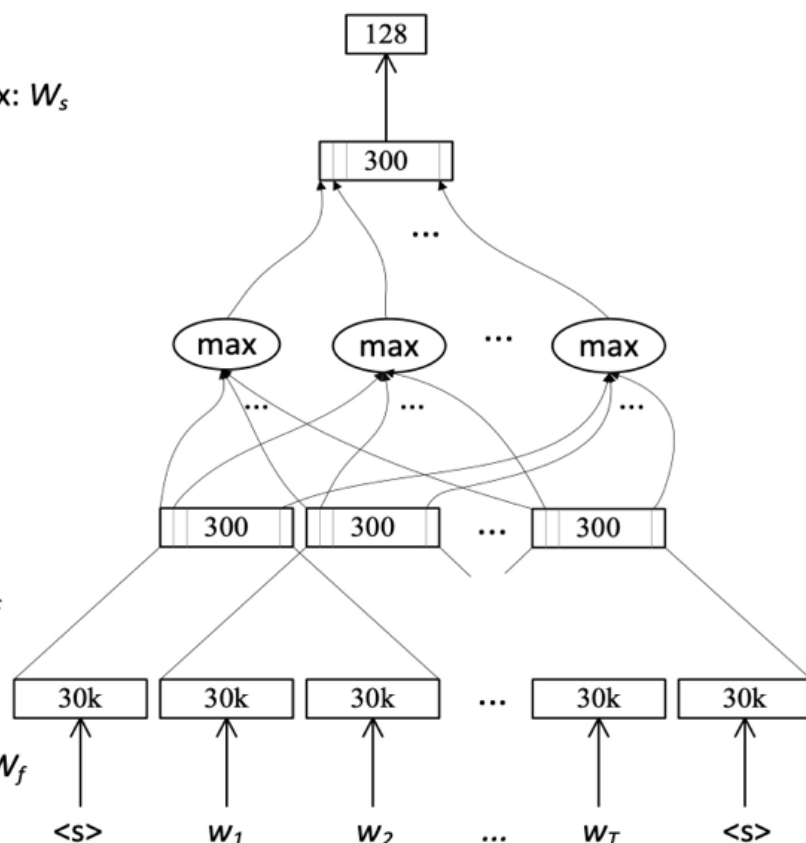
Convolutional layer: h_t

Convolution matrix: W_c

Word hashing layer: f_t

Word hashing matrix: W_f

Word sequence: x_t



Архитектура решения

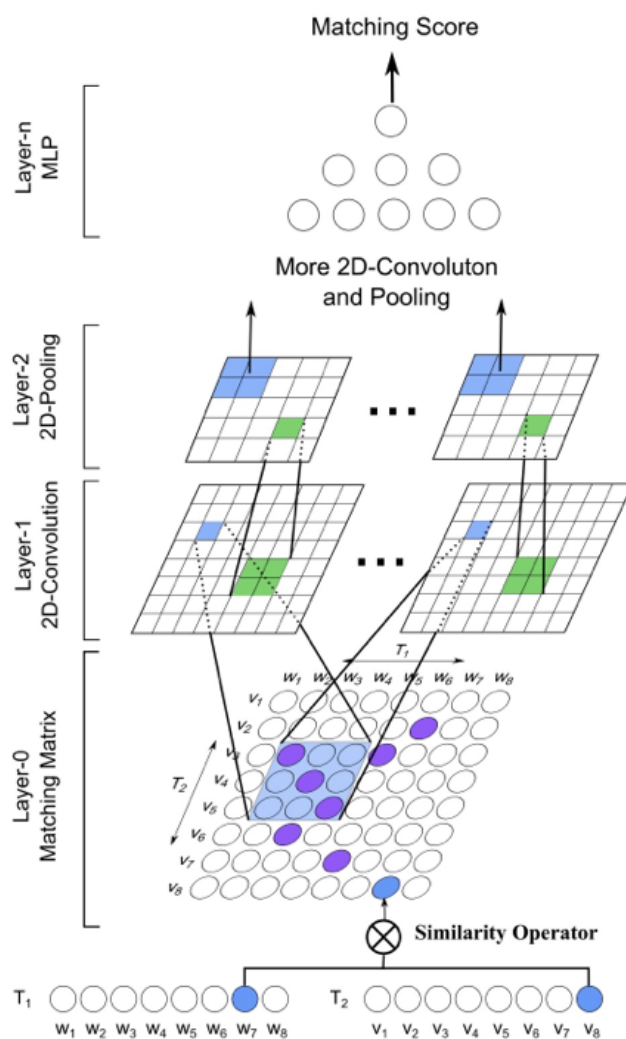
То есть теперь несколько отдельных слов разбиваются на триграммы, а вектор, характеризующий их, подаётся на вход **свёртке**, выходная размерность которой 300. Таким образом формируется некоторое количество эмбеддингов уже не всего предложения или текста, а **отдельных словосочетаний**.

Затем среди всех эмбеддингов по каждой из размерностей среди 300 берётся **максимальный элемент** — это равносильно выявлению наиболее ярко выраженных смысловых составляющих.

Таким образом, вне зависимости от длины текста, получается новый вектор размерности 300. **После проецирования** его в пространства меньшей размерности, например, 128, получается эмбеддинг всего текста.

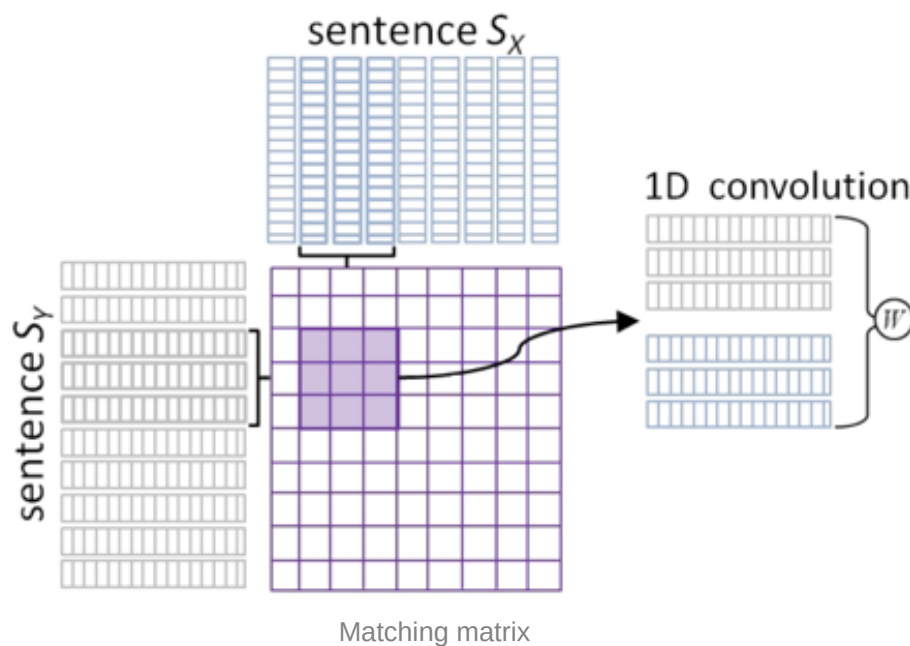
Почему это работает лучше: если есть **мусорные слова** или словосочетания, которые не выражают никакой мысли (а таких в текстах обычно большинство, они выполняют функцию связок), то их **промежуточные эмбеддинги будут околонулевыми**. Операция взятия максимума по каждой составляющей эмбеддинга либо никак не повлияет на результат, либо лишь слегка его изменит, если смысл в предложении не выражен.

> MatchPyramid



Архитектура MatchPyramid

Очень похожа на уже разобрannую [ARC-II](#), однако в [методе](#) на первом этапе не формируется матрица взаимодействия нескольких слов в скользящем окне. Свёртки выучивают взаимодействие "паттернов".



Matching matrix формируется с помощью лишь одной функции от двух эмбеддингов слов: одного из запроса и одного из документа.

Примеры:

- индикаторная функция,
- косинусные меры.

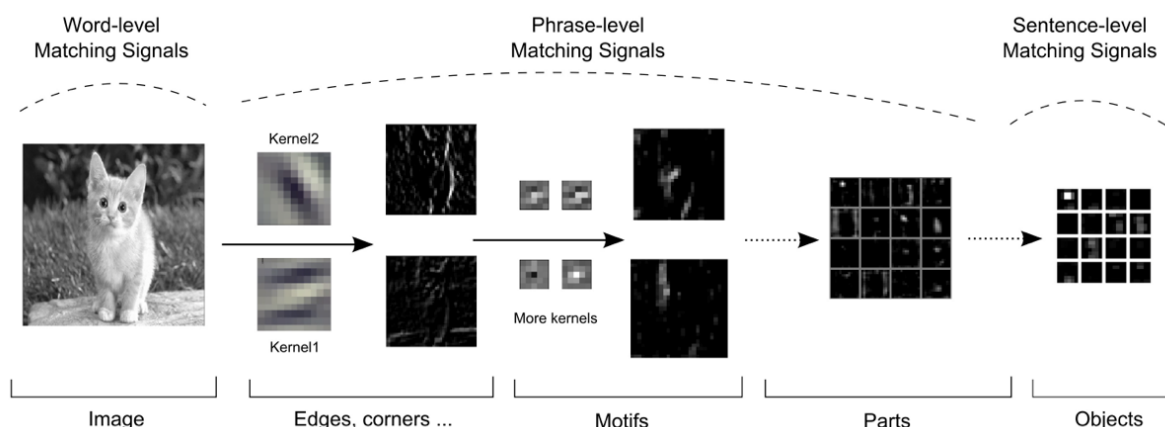
$$M_{ij} = \frac{\vec{\alpha}_i^T \vec{\beta}_j}{\|\vec{\alpha}_i\| \|\vec{\beta}_j\|}$$

$$M_{ij} = \vec{\alpha}_i^T \vec{\beta}_j$$

(Отличие в нормировке, но суть одна и та же)

Затем к полученной матрице взаимодействия слов применяются выучиваемые свёртки, пуллинги и в последствии **MLP** слой для определения финальной релевантности.

> Аналогия со сверточными сетями



Аналогия со свёрточными сетями, применительно к картинкам

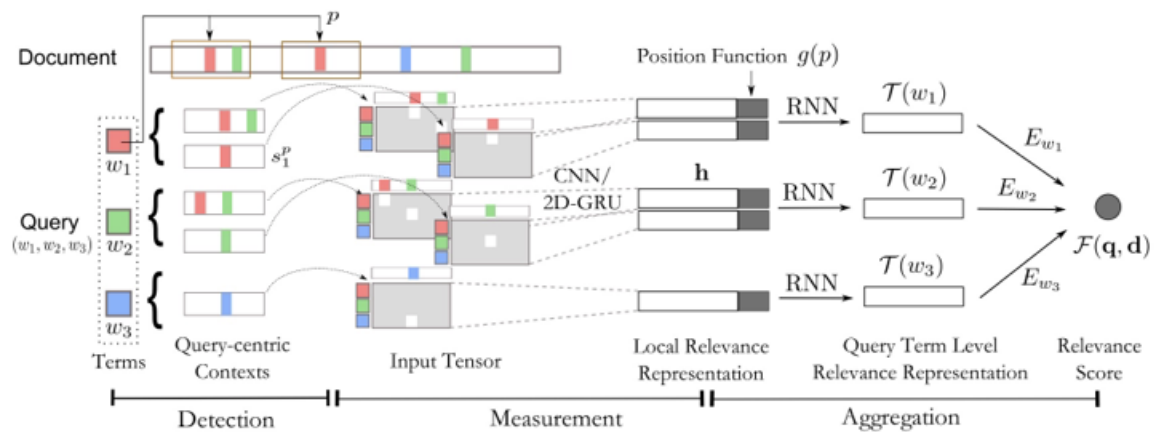
Авторы подхода проводят аналогию со свёрточными нейронными сетями для обработки изображений:

1. Первые свёртки из целой картинке выделяют **простейшие паттерны**, например углы и грани объектов.
2. Появляются всё более и более высокоуровневые представления отдельных частей изображения, а применительно к обработке текстов — **признаки на уровне фраз**.
3. Формируется **представление целой текстовой последовательности**, характеризующее текст с учётом всех аспектов на уровне отдельных фраз.

Для того чтобы это работало, внизу "пирамиды" должны быть самые простые признаки на уровне взаимодействия отдельных слов, а не сразу же интеракций между ними, как это было в **ARC-II**. В процессе прохождения через свёрточные слои с последовательной обработкой векторов **взаимодействия будут формироваться уже сами** в единой манере.

> Комбинирование методов и улучшение архитектур

> DeepRank



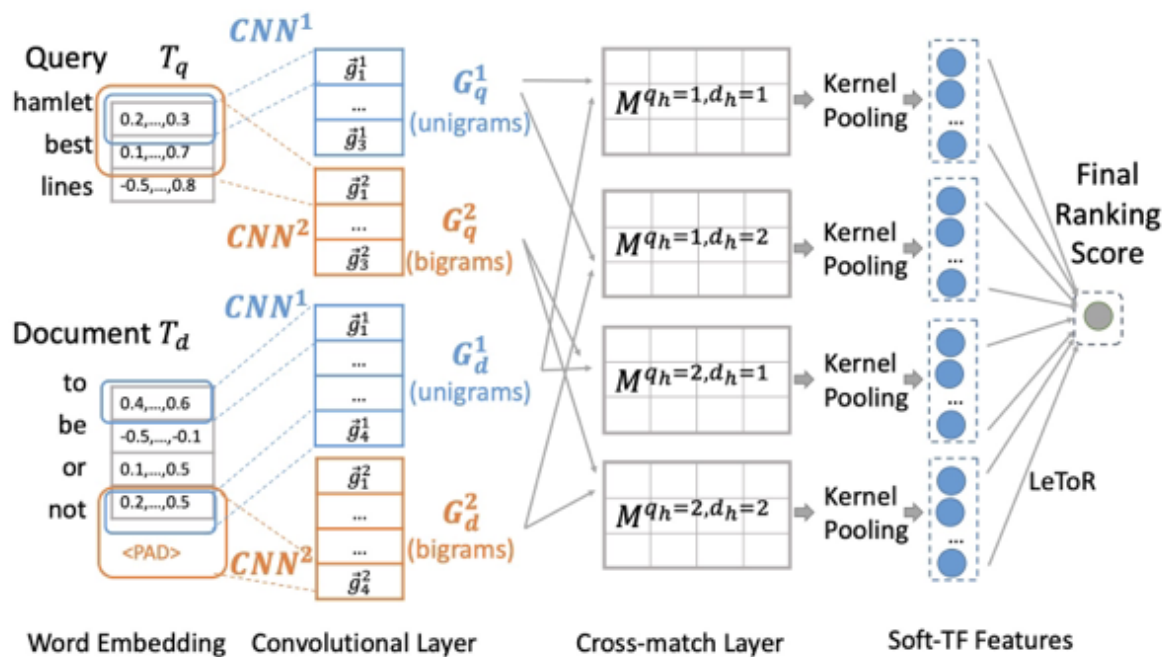
Архитектура DeepRank

В методе используется архитектура, состоящая из элементов, рассмотренных ранее:

- Формируется матрица взаимодействий слов из запроса и документа, т.е. **interaction matrix**.
- Поверх применяются свёртки, к результатам работы которых добавляется некоторая **позиционная информация**.
- Сэндвич из соединённых тензоров обрабатывается **RNN**.
- Финальное взвешивание по отдельным словам запроса формируется с помощью **gating network**.

Суть **позиционного кодирования перед взвешиванием** отдельных слов в том, что нейросеть при вынесении вердикта понимает, в каком порядке шли слова.

> Conv-KNRM



Архитектура Conv-KNRM

Отличие Conv-KNRM от KNRM:

- Как следует из названия, **вместо одного слова** некоторое скользящее окно с помощью свёрток формирует **новое представление для пар слов, троек слов и т.д.**
- **Вместо одной матрицы взаимодействий** формируются **попарные матрицы** взаимодействия результатов после свёрток, и уже к ним применяются те самые ядра, которые были рассмотрены на занятии с KNRM.

> Резюме

- Узнали про методы представления текста в виде векторов с помощью задачи языкового моделирования, **поняли интуицию**, лежащую за подходами обучения эмбеддингов.
- Рассмотрели сравнение векторов "на пальцах".
- **Хэширование** и **переход на N-граммы** позволяет уменьшить количество параметров в модели — так появился FastText
- На схожих идеях базируется DSSM — модель, обучаемая на предсказание релевантности по кликам (и самостоятельно выучивающая эмбеддинги,

соответствующие специфике задачи).

- Обучать эмбединги под свою задачу позволяет **широкий класс моделей** (от ранее рассмотренных **K-NRM** и **ARC-I / ARC-II** до **MatchPyramid**).
- Отталкиваясь от специфики конкретной задачи, можно комбинировать техники из разных методов, искать вдохновение в схожих задачах (в свёрточных сетях для изображений, в графовых моделях и т.д.).